A Memory Efficient Parallel Tridiagonal Solver

Travis M. Austin, Markus Berndt, and J. David Moulton (T-7); ***berndt@lanl.gov***

Large tridiagonal systems of linear equations appear in many numerical analysis applications. In our work, they arise in line relaxations needed by robust multigrid methods, such as the parallel BoxMG code [1], for structured grid problems. We present a new memory-efficient partitioning algorithm for the solution of diagonally dominant tridiagonal linear systems of equations that scales well on distributed-memory parallel computers. Its multilevel recursive design makes it well suited for distributed-memory parallel computers with very large numbers of processors.

On a serial computer, Gaussian elimination without pivoting can be used to solve a diagonally dominant tridiagonal system of linear equations in *O(N)* steps. This serial algorithm is commonly referred to as the Thomas algorithm [2]. Unfortunately, this algorithm is not well suited for parallel computers. The first parallel algorithm for the solution of tridiagonal systems was developed by Hockney and Golub. It is now usually referred to as cyclic reduction. Stone introduced his recursive doubling algorithm in 1973. Both cyclic reduction and recursive doubling are designed for fine-grained parallelism, where each processor owns exactly one row of the tridiagonal matrix. In 1981, Wang proposed a partitioning algorithm aimed at more coarse-grained parallel computation typical for shared memory clusters, where $N_P << N$. There has also been attention directed toward a parallel partitioning of the standard LU algorithm. In 1986, Sun et al. introduced the parallel partitioning LU algorithm that is very similar to Bondeli's divide and conquer algorithm. These algorithms, while well suited for problems distributed across a moderately large number of processors, do not scale well to very large numbers of processors.

Our algorithm can be described as a recursion with a partitioning algorithm as its basis. We begin by describing this partitioning algorithm. The tridiagonal linear system is assumed to be distributed across a large number of processors, such that each processor owns a contiguous number of rows. Each processor transforms its piece of the tridiagonal matrix into a matrix with a sparsity pattern such as

$$A_{\text{local}} = \begin{pmatrix} \times & \times & & & & \times & \\ & \times & \times & \times & & & \\ & & \times & \times & \times & & \\ & & & \times & \times & \times & \\ & \times & & & & \times & \times \end{pmatrix}.$$

Gathering the first and last rows (red) from every processor yields an interface system that is again tridiagonal and diagonally dominant. This interface system can be solved by gathering all equations to one processor, using the Thomas algorithm there, and then scattering the solution back to all N_P processors. Then the interface unknowns can be eliminated from the local systems, yielding N_P local tridiagonal systems (blue $\times$'s). These local systems can be solved efficiently by the Thomas algorithm and do not require any further communication.

Although this nonrecursive single-level approach scales reasonably well for moderate numbers of processors, it does not scale well for very large number of processors: gather and scatter operations typically scale linearly with the number of processors. Our remedy is to gather only pieces of the complete interface system to a subset of all processors, such that each of these subset-processors owns a contiguous piece of the interface system. Then we apply the partitioning outlined above to each piece of the interface system. This yields a lower-level interface system on the subset of processors. We proceed with further recursion if the subset of processors is sufficiently large, or solve the new interface system directly on one of the processors in the subset.

We describe our algorithm as memory-efficient because the partitioning step is organized such that the interface system is generated without overwriting the original tridiagonal system.

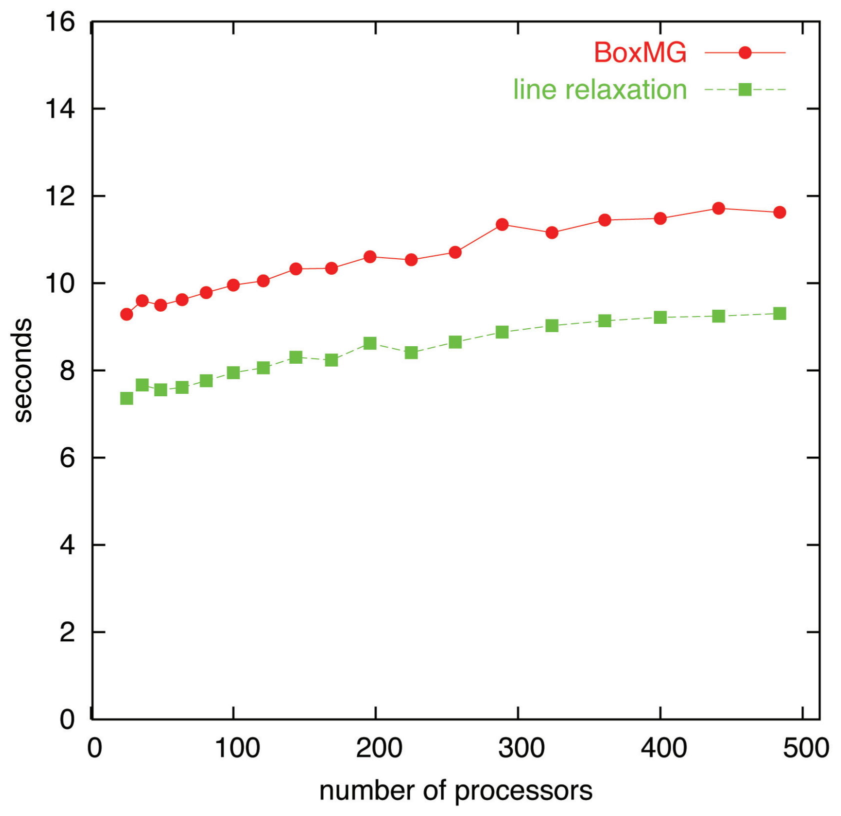

***Figure 1*— Timings for 20 V(1,1) BoxMG cycles with red-black line relaxation on square processor grids ranging from 2 × 2 to 22 × 22.**

For example, the figure shows timings for 20 $V(1,1)$ cycles with red-black line relaxation on a square processor grid with constant problem size on each processor. We observe very moderate growth in the solution time, which indicates good parallel scaling of our parallel-line relaxation algorithm.

The line solves are performed across up to 22 processors, without recursion in our algorithm. A careful study of the complexity of this nonrecursive algorithm indicates a linear dependency on the number of processors. Our recursive multilevel tridiagonal solver, would exhibit only logarithmic dependence on the number of processors. We will investigate the performance of this recursive algorithm in a future paper.

[1] T.M. Austin, M. Berndt, B.K. Bergen, J.E. Dendy, and J.D. Moulton, "Parallel, Scalable, and Robust Multigrid on Structured Grids," Los Alamos National Laboratory report LA-UR-03-9167 (2003).
[2] T.M. Austin, M. Berndt, and J.D. Moulton, "A Memory Efficient Parallel Line Solver," Los Alamos National Laboratory report LA-UR-04-4149 (2004) [submitted to SISC, 2004].

Los Alamos
NATIONAL LABORATORY
EST. 1943